# QBE: QLearning-Based Exploration of Android Applications

Presenter: Yavuz Koroglu

Yavuz Koroglu and Alper Sen

Dependable Systems Group (DSG)
Bogazici University, Istanbul, Turkey
http://depend.cmpe.boun.edu.tr
{yavuz.koroglu,alper.sen}@boun.edu.tr

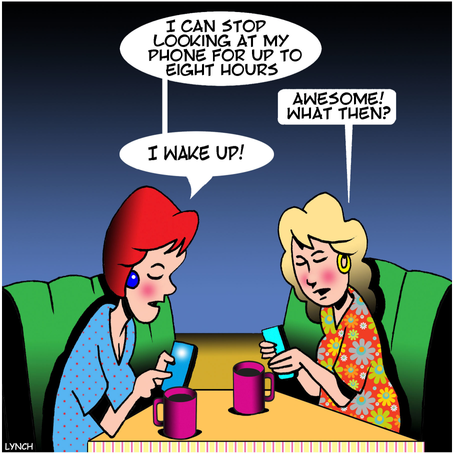
Ozlem Muslu, Ceyda Ulker, Yunus Mete, Tolga Tanriverdi, and Yunus Donmez

NETAS Telecommunications
Istanbul, Turkey
http://www.netas.com.tr/en/home-page/
yunusm@netas.com.tr

11th IEEE Conference on Software Testing, Validation and Verification (ICST'18)

# Overview

# Motivation



CartoonStock.com

## Mobile GUI Applications are Ubiquitous

- We use mobile phones often **(3 hours/day)**
- Mostly on mobile applications **(90% of the time spent)**

## Android Market is Growing

- **2.6 billion** mobile phone users

## Android has the Largest Share

- **82.8%** of all apps are for Android

# Publicly Available Automated Android GUI Testing Tools

- **Monkey**
- $A^3E$
- SwiftHand
- PUMA
- DynoDroid
- Sapienz

**Monkey**

Outperforms other tools in terms of

- **Coverage**
- **Crashes**

# Monkey

## Monkey

- Developed by **Google**
- Generates random
  1. **System events** and
  2. **GUI actions**
- **Built-in** (comes with the Android OS)

# Pros/Cons of Monkey

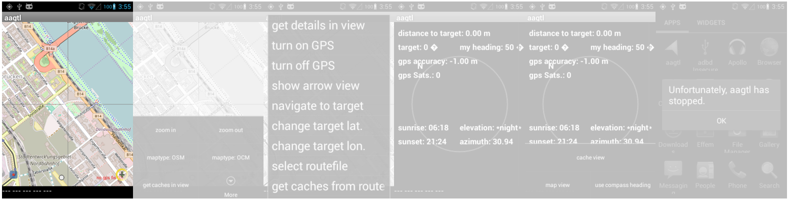
## Advantages

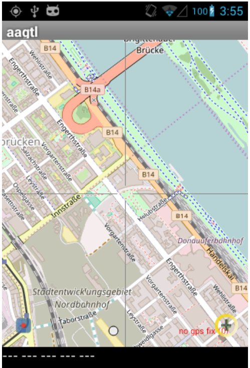
- **High Variety of Events**
  (Sensor, Navigation, System Events, Basic Gestures)
- **High Event Rate**
  (thousands of events per second)

## Disadvantages

- **Reproducibility Issues** (Poor Verifiability)
- Misses **Deep Crashes** and **Deep Activities**

# A Real Crash Found by None of the Other Tools



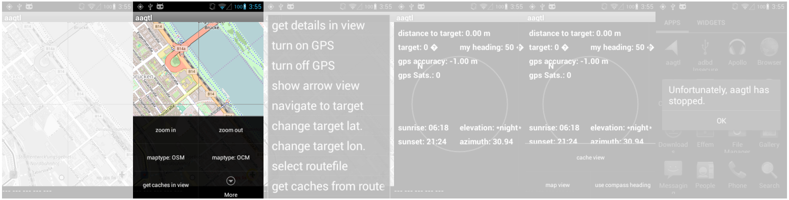



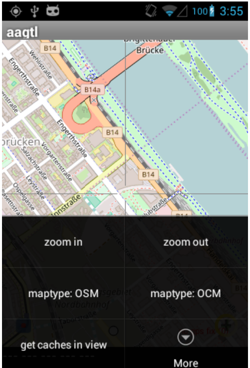

→ A GPS application.

→ Previous Actions: (1) **reinitialize**

→ Next Action: **menu**

# A Real Crash Found by None of the Other Tools



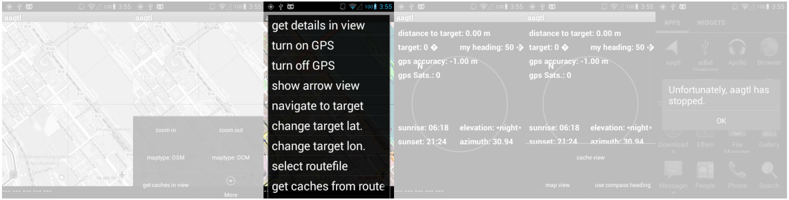

→ A GPS application.

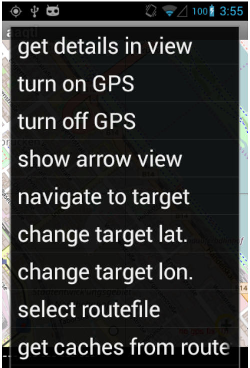
→ Previous Actions:

(1) **reinitialize**, (2) **menu**

→ Next Action: **click** More

# A Real Crash Found by None of the Other Tools



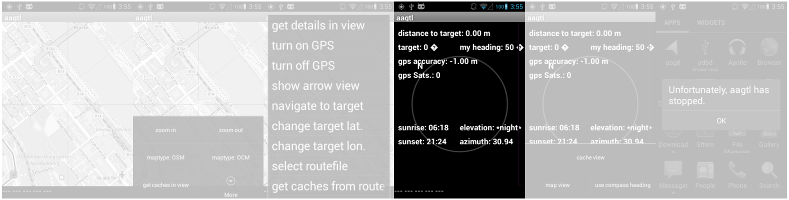

→ A GPS application.

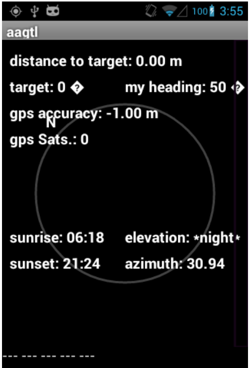
→ Previous Actions:

(1) **reinitialize**, (2) **menu**, (3) **click** More

→ Next Action: **click** show arrow view

# A Real Crash Found by None of the Other Tools



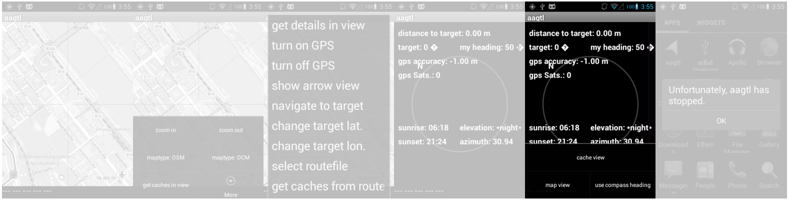

→ A GPS application.

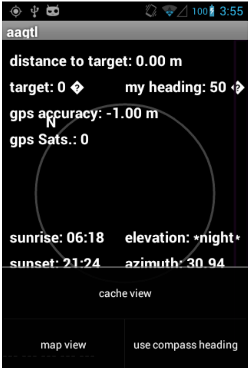
→ Previous Actions:

(1) **reinitialize**, (2) **menu**, (3) **click** More, (4) **click** show arrow view

→ Next Action: **menu**

# A Real Crash Found by None of the Other Tools



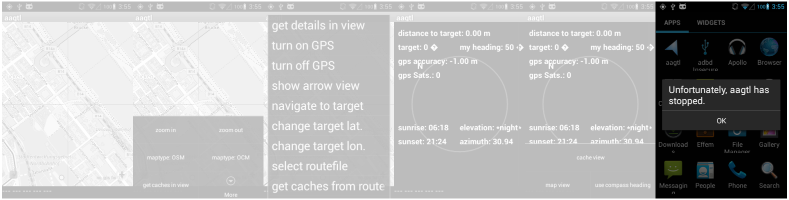

→ A GPS application.

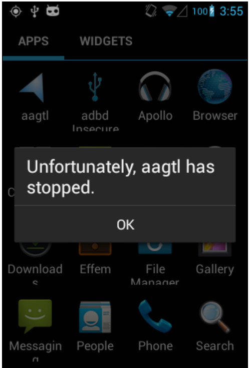
→ Previous Actions:

(1) **reinitialize**, (2) **menu**, (3) **click** More, (4) **click** show arrow view, (5) **menu**

→ Next Action: **click** cache view

# A Real Crash Found by None of the Other Tools




→ A GPS application.

→ Previous Actions:

(1) **reinitialize**, (2) **menu**, (3) **click** More, (4) **click** show arrow view, (5) **menu**, (6) **click** cache view

→ CRASH

→ **Monkey:** Probability of generating these actions in this order is very low.

→ **Others:** It takes a long time to systematically exhaust all possibilities.

# QLearning-Based Exploration (QBE) Overview

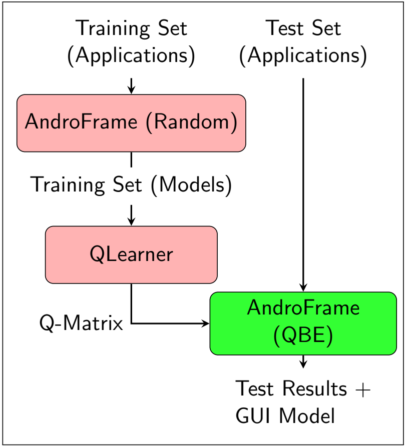



Figure: QLearning-Based Exploration (QBE) Overview

## Main Idea

- To **learn** the best actions in similar states.

## Main Flow

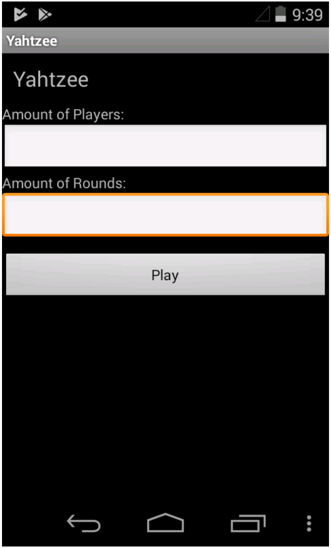
1. **Explore** the training set (with random exploration)
2. **Generate** GUI Models
3. **Learn** the best transitions
4. **Direct** the testing process (use the learned model)

# Model-Based GUI Testing of Android Applications

**In general,**

- Most applications do **NOT** have a model
- Learn the application model **dynamically**
- The model is an **Extended Labeled Transition System (ELTS)** where
  1. **Nodes** are GUI **states**.
  2. **Edges** are transitions via GUI **actions**.

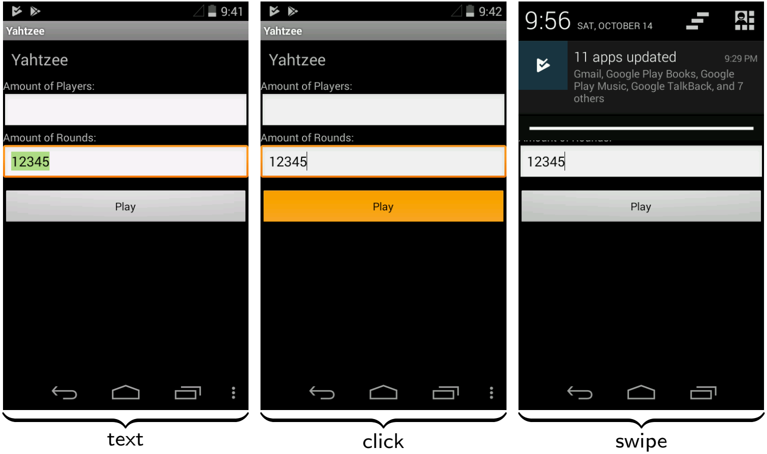
# GUI State




1. **Java Package Name**
2. **Activity Name**
   (An activity roughly corresponds to an Android screen)
3. **Contextual Attributes**
   (WiFi, Orientation etc.)
4. **GUI Components (widgets)**
   on the screen

# GUI Action

User-triggered events: **text, click, swipe** etc.



# AndroFrame: Automated Test Generation Framework

## What is AndroFrame?

It is a

- **Fully-automated**,
- **Black-box**,
- **Modular**,
- **Automata Learning**

replayable test case generation framework.

## Important

- We build QBE on top of AndroFrame.

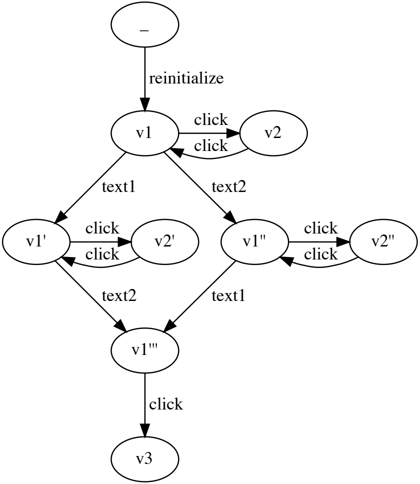
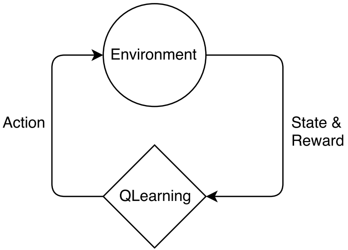



Figure: Example Model of the Yahtzee App

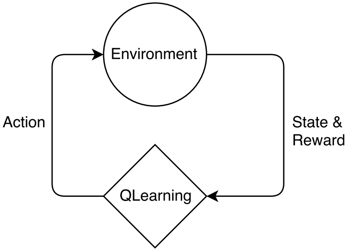
# QLearner




## Main Idea

- QLearner observes
  1. The current **state** and
  2. The latest **reward**
- QLearner decides on
  1. An action

# QLearner




## Main Idea

- QLearner observes
  1. The current **state** and
  2. The latest **reward**
- QLearner decides on
  1. An action

## Q-Matrix

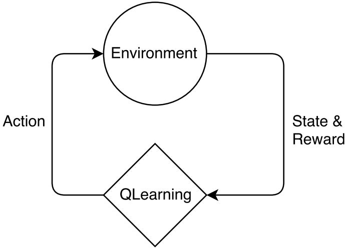
A matrix of values where

- Rows are **states** and
- Columns are **actions**.

## Q-Value

- Cells in the Q-Matrix.
- Associated with a **state-action pair**.
- **Expectancy** of the action **getting a reward** in the next state.

# QLearner




## Main Idea

- QLearner observes
  1. The current **state** and
  2. The latest **reward**
- QLearner decides on
  1. An action

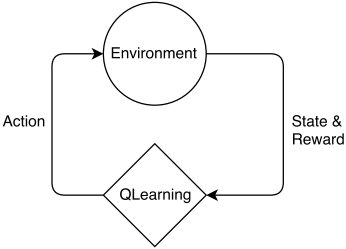
## Example

| | click | text |
|---|---|---|
| s1 | 1 | 0 |
| s2 | 0 | 0 |
| s3 | 0.17 | 0.83 |

## Important

- All rows add up to 1 (except unvisited states)
- At **s1**, always **click**
- At **s2**, no knowledge (all 0s)
- At **s3**, mostly **text**

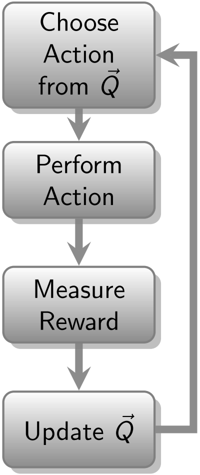
# QLearner




## Main Idea

- QLearner observes
  1. The current **state** and
  2. The latest **reward**
- QLearner decides on
  1. An action

Initially, $\vec{Q} = 0$.



# QLearning: Standard Updates

$$\underbrace{\vec{Q}[s,a]}_{\substack{\text{Next}\\ \text{Q-Matrix}}} \leftarrow \underbrace{\vec{Q}[s,a]}_{\substack{\text{Previous}\\ \text{Q-Matrix}}} + \underbrace{\vec{N}[s,a]^{-1}}_{\substack{\text{History}\\ \text{Matrix}}} \left( \underbrace{o(v,z)}_{\substack{\text{Objective}\\ \text{Function}}} + \underbrace{\gamma \vec{Q}[s',a']}_{\substack{\text{Future}\\ \text{Expectancy}}} - \underbrace{\vec{Q}[s,a]}_{\substack{\text{Previous}\\ \text{Q-Matrix}}} \right)$$

## Definitions

- **History Matrix:** A **running count** of previous updates on each $\vec{Q}[s,a]$.
- **Objective Function:** Denotes the **reward**. 1 if the goal is satisfied, 0 otherwise.
- **Future Expectancy:** Allows future rewards to be **propagated** along an execution path.
- **Discount Factor ($\gamma$):** A value btw 0 and 1 to **decrease the future expectancy** as the path gets longer.

# Illustrative Example: How QLearning Works

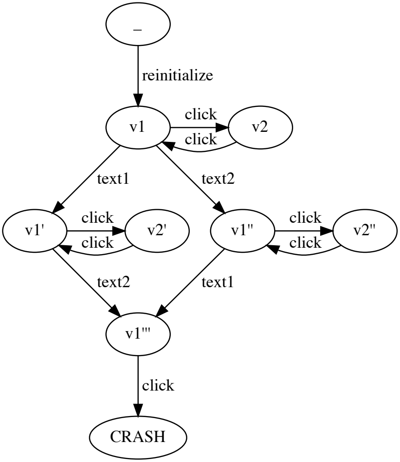



Figure: GUI Model of the Yahtzee App

## Without Abstraction

- 7 application states (excluding "_" and "CRASH")
- 11 state-action pairs (excluding "reinitialize")
- Would be **too large** in real scenarios.

## Similar States

- Cosine Similarity $> 0.95$
  1. $v1, v1', v1'', v1'''$ and
  2. $v2, v2', v2''$

# Illustrative Example: How QLearning Works

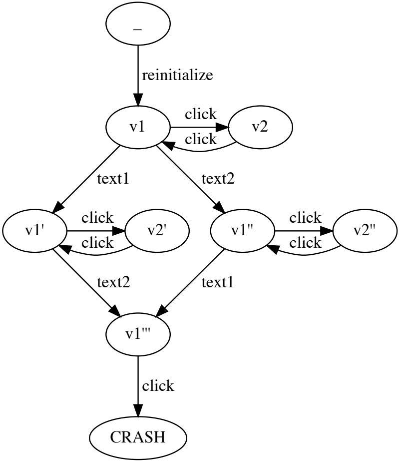



Figure: GUI Model of the Yahtzee App

## Let's Abstract

- States (2 state types)
  1. $s1 = \{v1, v1', v1'', v1'''\}$
  2. $s2 = \{v2, v2', v2''\}$
- Actions (2 action types)
  1. click
  2. text
- We get a 2 by 2 matrix: $\vec{Q}[s, a]$

# Illustrative Example: How QLearning Works

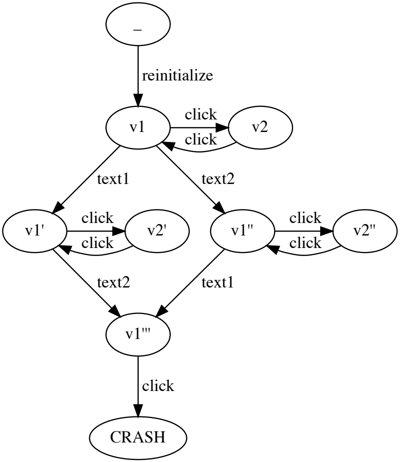



Figure: GUI Model of the Yahtzee App

## Initial Q-Matrix

| | click | text |
|---|---|---|
| s1 | 0 | 0 |
| s2 | 0 | 0 |

The only way to update Q-values is to

- **Get a reward**

# Illustrative Example: How QLearning Works

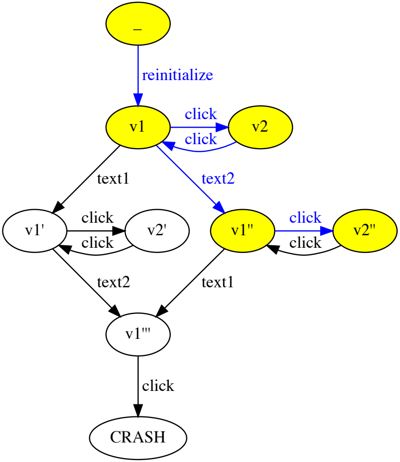



Figure: GUI Model of the Yahtzee App

## New Q-Matrix

| | click | text |
|---|---|---|
| s1 | 0 | 0 |
| s2 | 0 | 0 |

Test Case: $v1, v2, v1, v1'', v2''$

- No **rewards**, no **updates**.

# Illustrative Example: How QLearning Works

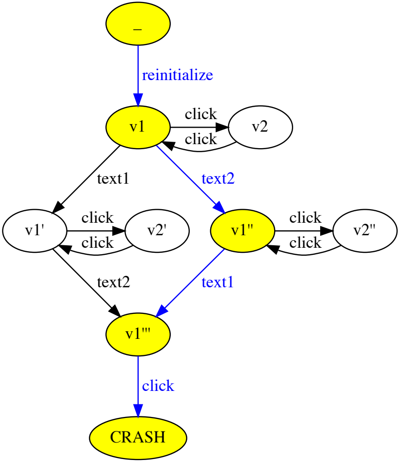



Figure: GUI Model of the Yahtzee App

## New Q-Matrix

| | click | text |
|---|---|---|
| s1 | 1 | 0 |
| s2 | 0 | 0 |

Test Case: $v1, v1'', v1''', CRASH$

- Learns the last transition first.

# Illustrative Example: How QLearning Works

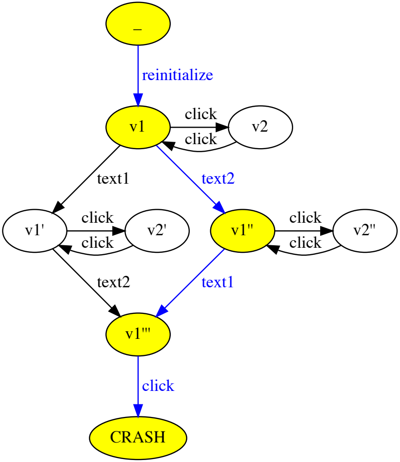



Figure: GUI Model of the Yahtzee App

## New Q-Matrix

| | click | text |
|---|---|---|
| s1 | .53 | .47 |
| s2 | 0 | 0 |

Test Case: $v1, v1'', v1''', CRASH$ (again)

- Now, $v1'' \rightarrow v1'''$ also gets Q-value, due to **future value**.

# Illustrative Example: How QLearning Works

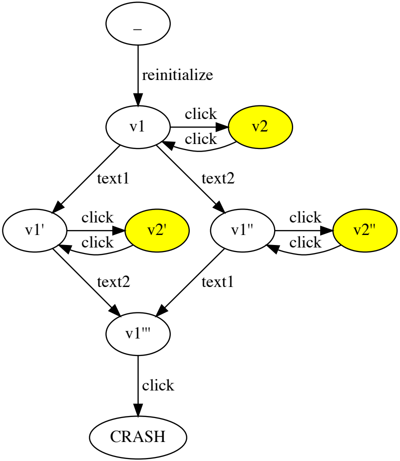



Figure: GUI Model of the Yahtzee App

## Converged Q-Matrix

| | click | text |
|---|---|---|
| s1 | .57 | .43 |
| s2 | 1 | 0 |

- At all s2 states ($v2, v2', v2''$), QBE always **clicks**.

# Reward (Objective) Function

Two reward functions
($v$: Current State, $z$: GUI Action, $v'$: Next State)

## 1. Crash Detection

$$o(v,z) = \begin{cases} 1 & v' \text{ is a CRASH state} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

## 2. Activity Coverage Increase

$$o(v,z) = \begin{cases} 1 & v' \text{ belongs to a new Activity} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

# Common Evaluation Criteria

## Number of Distinct Crashes

- **Parse** the Android logs (Common technique)
- **Stack traces for exceptions** are also in these logs
- Do NOT count the same stack trace more than once

## Activity Coverage

- A **high level metric** that is necessary to claim a high coverage of functionality (# Explored Activities / # All Activities)

## Instruction Coverage

- A **low level metric** that shows the amount of code utilization (# Explored Instructions / # All Instructions)

# Experimental Setup

- 14 x Android-x86 VirtualBox guests (with Android 4.4.r5)
- 300 Android applications randomly selected from F-Droid benchmarks
  - 200 training and 100 test applications
- 10 minutes for each application.
- Implemented 4 Strategies in AndroFrame,
  1. Random Exploration (RE)
  2. Depth-First Exploration (DFE)
  3. Activity-Based QBE (QBEa)
     - Reward function is **Activity Coverage Increase**.
  4. Crash-Based QBE (QBEc)
     - Reward function is **Crash Detection**.

# Experimental Results

Table: Experimental Results over 10 minutes

| | Tool | Activity (%) | Instr. (%) | #Crashes |
|---|---|---|---|---|
| AndroFrame | Activity-Based QBE (QBEa) | **78** | 40 | 7.8 |
| | Crash-Based QBE (QBEc) | 65 | 32 | **12.6** |
| | Depth-First Exploration (DFE) | 63 | 34 | 3 |
| | Random Exploration (RE) | 58 | 30 | 3.2 |
| Others | DynoDroid | 50 | 35 | 5.2 |
| | $A^3E$ | 41 | 17 | 8 |
| | Monkey | 60 | 30 | **9** |
| | PUMA | 64 | 32 | 6 |
| | Sapienz | 76 | **44** | 4 |
| | SwiftHand | 40 | 19 | 0 |

**QBEa** has the best activity coverage.

# Experimental Results

Table: Experimental Results over 10 minutes

| | Tool | Activity (%) | Instr. (%) | #Crashes |
|---|---|---|---|---|
| AndroFrame | Activity-Based QBE (QBEa) | **78** | 40 | 7.8 |
| | Crash-Based QBE (QBEc) | 65 | 32 | **12.6** |
| | Depth-First Exploration (DFE) | 63 | 34 | 3 |
| | Random Exploration (RE) | 58 | 30 | 3.2 |
| Others | DynoDroid | 50 | 35 | 5.2 |
| | $A^3E$ | 41 | 17 | 8 |
| | Monkey | 60 | 30 | **9** |
| | PUMA | 64 | 32 | 6 |
| | Sapienz | 76 | **44** | 4 |
| | SwiftHand | 40 | 19 | 0 |

**Sapienz** has better code coverage.

# Experimental Results

Table: Experimental Results over 10 minutes

| | Tool | Activity (%) | Instr. (%) | #Crashes |
|---|---|---|---|---|
| AndroFrame | Activity-Based QBE (QBEa) | **78** | 40 | 7.8 |
| | Crash-Based QBE (QBEc) | 65 | 32 | **12.6** |
| | Depth-First Exploration (DFE) | 63 | 34 | 3 |
| | Random Exploration (RE) | 58 | 30 | 3.2 |
| Others | DynoDroid | 50 | 35 | 5.2 |
| | $A^3E$ | 41 | 17 | 8 |
| | Monkey | 60 | 30 | **9** |
| | PUMA | 64 | 32 | 6 |
| | Sapienz | 76 | **44** | 4 |
| | SwiftHand | 40 | 19 | 0 |

**QBEc** detects the highest number of crashes.

# Experimental Results

Table: Experimental Results over 10 minutes

| | Tool | Activity (%) | Instr. (%) | #Crashes |
|---|---|---|---|---|
| AndroFrame | Activity-Based QBE (QBEa) | **78** | 40 | 7.8 |
| | Crash-Based QBE (QBEc) | 65 | 32 | **12.6** |
| | Depth-First Exploration (DFE) | 63 | 34 | 3 |
| | Random Exploration (RE) | 58 | 30 | 3.2 |
| Others | DynoDroid | 50 | 35 | 5.2 |
| | $A^3E$ | 41 | 17 | 8 |
| | Monkey | 60 | 30 | **9** |
| | PUMA | 64 | 32 | 6 |
| | Sapienz | 76 | **44** | 4 |
| | SwiftHand | 40 | 19 | 0 |

QBE is successful at **coverage** and **crash detection**

# Conclusions and Future Work

## Conclusions

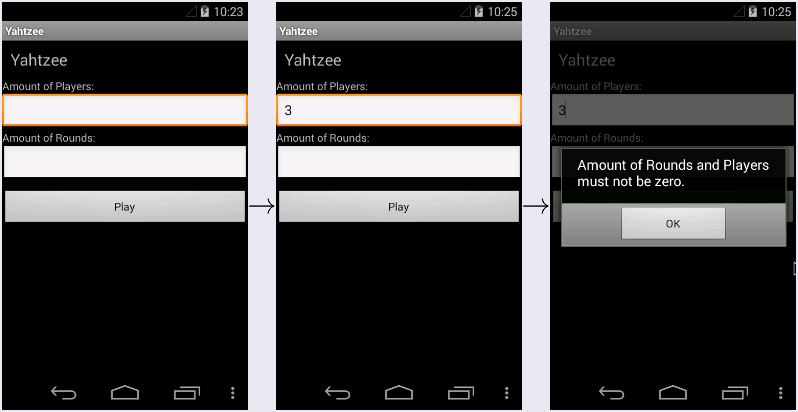
- QLearning-Based Exploration (QBE) for Model Based GUI Testing of Android Applications
- Experiments on 100 applications. QBE
  1. Achieves **the highest activity coverage** and
  2. Finds **the most distinct crashes**.

## Future Work

- **More reward functions**, e.g. code coverage increase.
- Improve **abstraction functions**.
- **Online QLearning** for app-specific patterns.
- Use other Machine Learning techniques to improve testing.

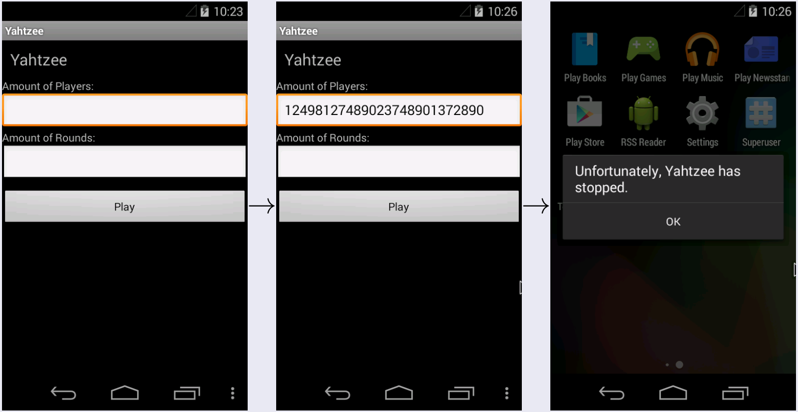
# TCM: Test Case Mutation to Improve Crash Detection in Android, Published @ FASE'18

## An Automatically Generated Test Case



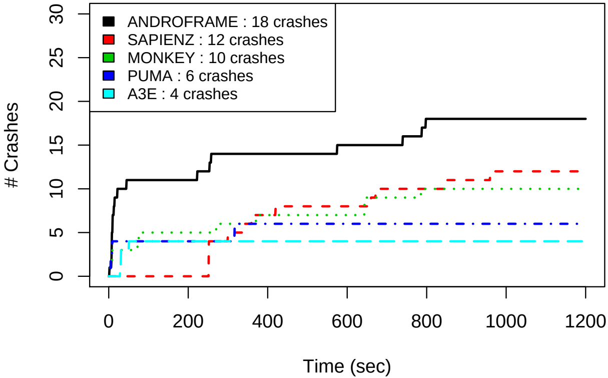
# TCM: Test Case Mutation to Improve Crash Detection in Android, Published @ FASE'18

## Mutated Test Case



Thank You! Any Questions?

# Appendix A: Recent Results Across Time




Shows that AndroFrame finds distinct crashes from **very early on**.

# Appendix B: Table of GUI Actions

Table: List of GUI Actions for our Automated Testing Tool

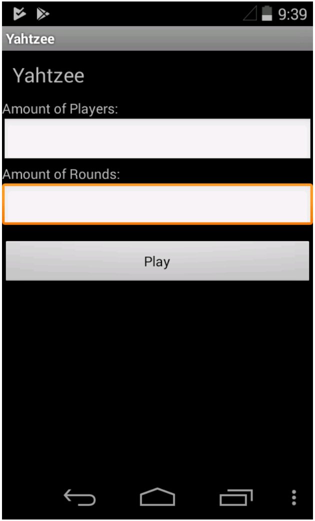
| **Non-contextual** | Param1 | Param2 | Param3 | Param4 | Param5 |
|---|---|---|---|---|---|
| click | x | y | - | - | - |
| longclick | x | y | - | - | - |
| text | x | y | string | - | - |
| swipe | x1 | y1 | x2 | y2 | duration |
| menu | - | - | - | - | - |
| back | - | - | - | - | - |

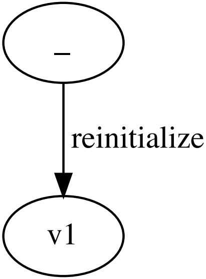
| **Contextual** | Parameters |
|---|---|
| connectivity | on/off/toggle |
| bluetooth | on/off/toggle |
| location | gps/gps&network/off/toggle |
| planemode | on/off/toggle |
| doze | on/off/toggle |

| **Special** | Param1 | Param2 | Param3 | Param4 | Param5 |
|---|---|---|---|---|---|
| reinit | package | activity | - | - | - |

# Appendix C: Automatic Generation of GUI Models Example

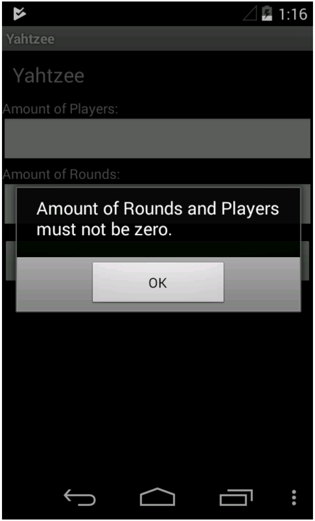
**Action:** reinitialize com.tum.yahtzee MainActivity

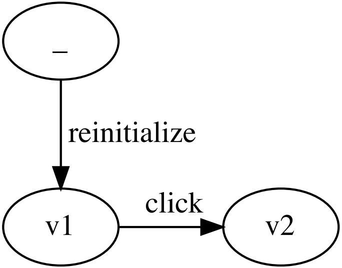


# Appendix C: Automatic Generation of GUI Models Example

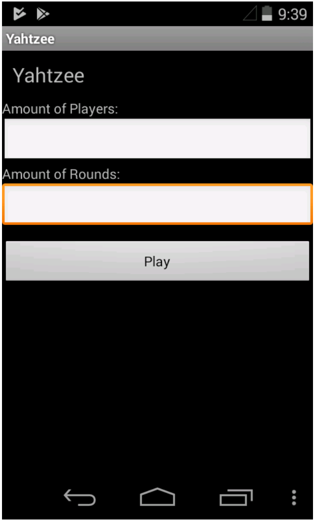
**Action:** click 200 390 (click play)

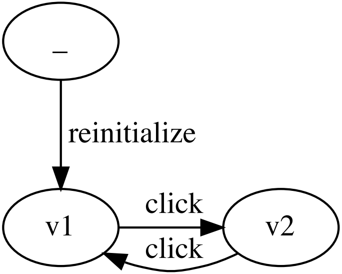


# Appendix C: Automatic Generation of GUI Models Example

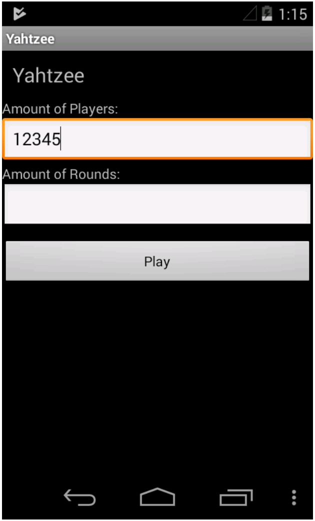
**Action:** click 200 410 (click ok)

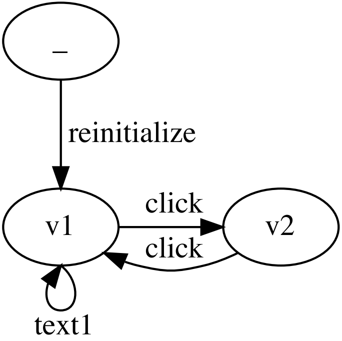


# Appendix C: Automatic Generation of GUI Models Example

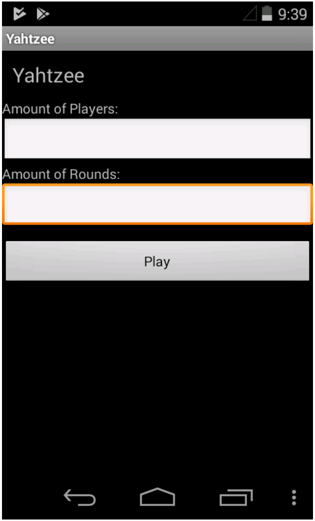
**Action:** text 200 270 12345 (text1)

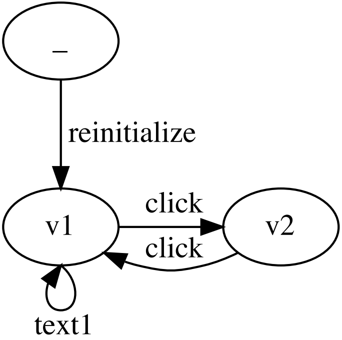


# Appendix C: Automatic Generation of GUI Models Example

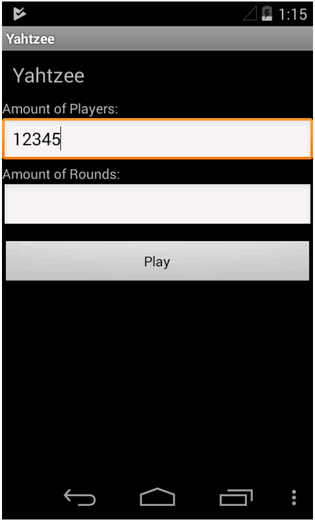
**Action:** reinitialize com.tum.yahtzee MainActivity

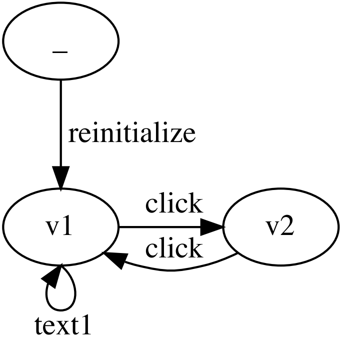


# Appendix C: Automatic Generation of GUI Models Example

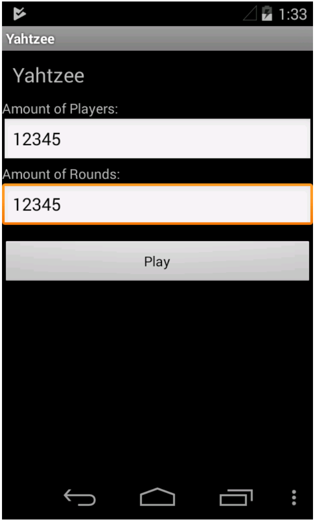
**Action:** text 200 270 12345 (text1)

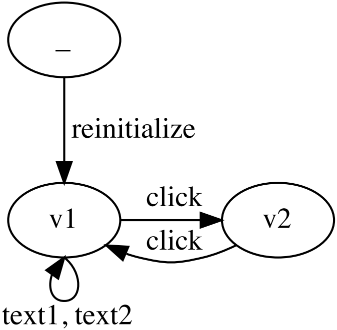


# Appendix C: Automatic Generation of GUI Models Example

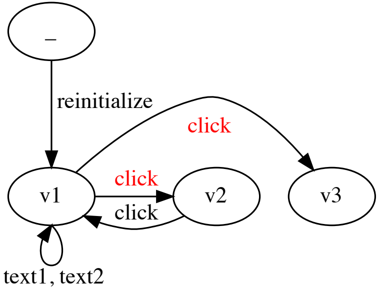
**Action:** text 200 330 12345 (text2)



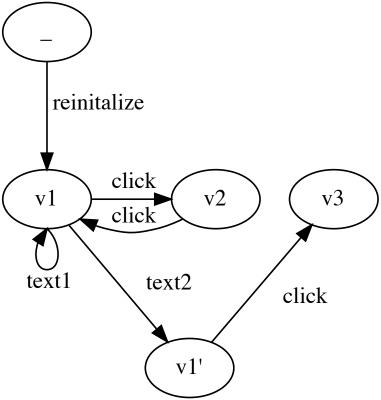
# Appendix C: Automatic Generation of GUI Models Example

**Action:** click 200 390 (click play)

# Appendix C: Automatic Generation of GUI Models Example

**Action:** click 200 390 (click play)

# Appendix D: Abstraction Functions in the Paper

$$\beta(v) = \begin{cases} 1, & |\lambda(v)| \leq 1 \\ 2, & |\lambda(v)| \leq 3 \\ 3, & |\lambda(v)| \leq 8 \\ 4, & |\lambda(v)| \leq 15 \\ 5, & |\lambda(v)| > 15 \end{cases} \quad \alpha(z) = \begin{cases} 1, & z \text{ is a } \textit{menu} \\ 2, & z \text{ is a } \textit{back} \\ 3, & z \text{ is a } \textit{click} \\ 4, & z \text{ is a } \textit{longclick} \\ 5, & z \text{ is a } \textit{text} \\ 6, & z \text{ is a } \textit{swipe} \\ 7, & z \text{ is a } \textit{contextual} \end{cases} \tag{3}$$

- $\lambda(v)$ denotes the **set of enabled actions** in the state $v$.
- $\beta(v)$ and $\alpha(z)$ abstract **states** and **actions**, respectively.
- These abstraction functions are simple and arbitrary. They are **open to improvement**.

# Appendix E: Benchmark Characteristics

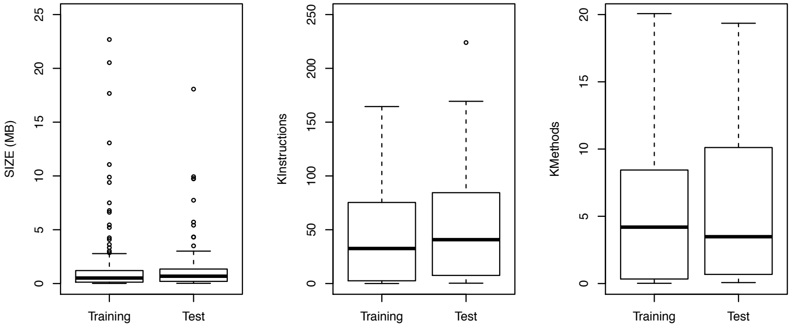



Figure: Characteristics of Training and Test Sets

## Between

- 0.01-25 MB, 1000-250000 instructions, and 10-20000 methods